Docket No.: 09386/100M230-US1
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Jim Belcher et al.

Application No.: 10/712,648 Confirmation No.: 8968

Filed: November 12, 2003 Art Unit: 2164

For: REMOTE INTELLIGENT CONTENT AUTHORING AND CONVERSION SYSTEM Examiner: B. M. Ortiz

DECLARATION UNDER 37 C.F.R. § 1.131

Dear Sir:

Jim BELCHER, Howard SOROKA, and Shari YOUNG being duly sworn, depose and say:

1. We are inventors of the patent application identified above, the inventors of the subject matter described and claimed therein, and over 21 years of age.

2. We have read the claims 1-3, as filed and amended in the Response dated November 21, 2006, and attached to this Declaration as Exhibit A.

3. Prior to February 21, 2002, the effective date of the Kim reference (U.S. Patent Publication No. 2004/0139483), we had completed our invention as described and claimed in the subject application in this country, a NAFTA country, or a WTO member country.

4. As evidence that our work antedates Kim, we refer to the Source Code, attached hereto as Exhibit B. The Source Code comprises at least 7 different sub-routines, created prior to the effective date of Kim. Dates, portions of the sub-routine that do not pertain to the subject matter claimed, and certain other proprietary disclosures appearing in Exhibit B have been redacted. We declare that this document, Exhibit B, was created before February 21, 2002.

5. We declare that the invention was reduced to practice before February 21, 2002, in that the Source Code disclosed in Exhibit B was tested and found to run successfully before the effective date of Kim.

6. Additionally, we refer to a series of internal e-mails between the inventors and the programmer or between members of the Assignee discussing aspects of the invention, including the Universal Resource Locator (URL) redirect function (attached hereto as Exhibit C1-C5, collectively "Exhibit C"). We declare that these documents, Exhibit C, were created before February 21, 2002.

7. Claims 1-3 are pending in the application. With respect to the subject matter of the claims, the Source Code discloses an identification system to identify the plurality of content files and a relation system to relate related content files. *See*, Exhibit B, pages 1-7. The Source Code also discloses a query system to request content files from a server, a reference database to store information about the plurality of content files and a collection system to convert the plurality of content files into at least one collection file. *See*, Exhibit B, pages 7-

13. Additionally, the Source Code shows a conversion module to covert the at least one collection file into a master copy, wherein the master copy is in one of a plurality of formats. *See*, Exhibit B, pages 14-15. *See*, for example, the elements of claim 1.

8. Regarding the subject matter of the claims, the Source Code discloses the steps of, with respect to claim 3, transmitting a plurality of content files in various formats to an authoring system, and identifying, by the authoring system, the plurality of content files. *See*, Exhibit B, pages 1-5, and 14-15. The Source Code also discloses the steps of relating, by the authoring system, the plurality of content files and storing the related content files together. *See*, Exhibit B, pages 5-7 and 15-16.

9. Additionally, the Source Code discloses querying a server for additional content files. The code that queries a server for additional content files is disclosed in Exhibit B, page 7. The below code calls a subroutine that moves and extracts files from one location on the server to another location:

```
# extract the ecd template files
    print "Extracting DataPlay template archive ...", br;
    ExtractTemplateFiles ($tarRootPath, $dptartemplate);
```

The following subroutine, in Exhibit B, page 9, extracts a whole set of directories and files from a single archive and saves them to the server from which the code is being served:

```
sub ExtractTemplateFiles {
    my $tarRootPath = shift @_;
    my $tartemplate = shift @_;

    # chdir to the destination dir
    chdir($tarRootPath) or die "Can't chdir $tarRootPath: $!";
```

```
  # read the source tar file
#   print "about to read $tartemplate", br;
   my $tar  = new Archive::Tar($tartemplate);

   # write the source tar files to the new directory
   my @files = $tar->list_files;
   $tar->extract(@files);
}
```

Also see, Exhibit B, pages 7-9, specifically on page 8 note the subroutine identifiers for populating album and track directories.

10. Also, the Source Code supports populating a reference database with information related to the plurality of content files. *See*, Exhibit B, pages 9-12. Further, the Source Code discloses the steps of creating, by a user, at least one collection file including content files and producing, by the authoring system, a master copy of the collection file in a format specified by the user. The code in Exhibit B, pages 12-13 (subroutine CreateDataPlayOutput) creates and populates the file structure for a formatted disc, compiles that file structure into a single formatted file, and displays a link on a web page for the user to download the single file. Further, the code in Exhibit B, page 15 (subroutine CreateECDOutput) extracts a file structure from a single file, creates and saves a control file defining the content on the disc, then creates a single file with the file structure and control file. The subroutine then displays a link on a web page that the user can click to download the single file.

11. Exhibit C discloses a URL configuration system to identify a URL entered by a user into at least one collection file and to communicate with a redirect server to activate the URL, and

the steps of determining if a URL is in the collection file and communicating with a redirect server to activate the URL. Exhibits C1-C5 show the conception and development of the above concept. *See*, claims 1 and 3.

12. I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful statements and the like so made are punishable by fine or imprisonment or both under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

James Belcher — 6-29-07
Jim BELCHER — Dated

Howard Sorota — 6/29/07
Howard SOROKA — Dated

Shari YOUNG — Dated

EXHIBIT A

EXHIBIT A

1. A system for remote authoring of content, comprising:

a plurality of content files; and

an authoring system storing the plurality of content files, comprising;

an identification system to identify the plurality of content files;

a relation system to relate related content files;

a query system to request content files from a server;

a reference database to store information about the plurality of content files;

a collection system to convert the plurality of content files into at least one collection file;

a conversion module to covert the at least one collection file into a master copy, wherein the master copy is in one of a plurality of formats; and

a URL configuration system to identify a URL entered by a user into the at least one collection file and to communicate with a redirect server to activate the URL.

2. The system of claim 1, further comprising a remote workstation for a user to access the authoring system.

3. A method of remotely authoring content, comprising:

transmitting a plurality of content files in various formats to an authoring system;

identifying, by the authoring system, the plurality of content files;

relating, by the authoring system, the plurality of content files;

storing the related content files together;

querying a server for additional content files;

populating a reference database with information related to the plurality of content files;

creating, by a user, at least one collection file including content files;

producing, by the authoring system, a master copy of the collection file in a format specified by the user;

determining if a URL is in the collection file;

communicating with a redirect server to activate the URL.

Docket No.: 09386/100M230-US1
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Jim Belcher et al.

Application No.: 10/712,648 Confirmation No.: 8968

Filed: November 12, 2003 Art Unit: 2164

For: REMOTE INTELLIGENT CONTENT AUTHORING AND CONVERSION SYSTEM Examiner: B. M. Ortiz

DECLARATION UNDER 37 C.F.R. § 1.131

Dear Sir:

Jim BELCHER, Howard SOROKA, and Shari YOUNG being duly sworn, depose and say:

1. We are inventors of the patent application identified above, the inventors of the subject matter described and claimed therein, and over 21 years of age.

2. We have read the claims 1-3, as filed and amended in the Response dated November 21, 2006, and attached to this Declaration as Exhibit A.

3. Prior to February 21, 2002, the effective date of the Kim reference (U.S. Patent Publication No. 2004/0139483), we had completed our invention as described and claimed in the subject application in this country, a NAFTA country, or a WTO member country.

4. As evidence that our work antedates Kim, we refer to the Source Code, attached hereto as Exhibit B. The Source Code comprises at least 7 different sub-routines, created prior to the effective date of Kim. Dates, portions of the sub-routine that do not pertain to the subject matter claimed, and certain other proprietary disclosures appearing in Exhibit B have been redacted. We declare that this document, Exhibit B, was created before February 21, 2002.

5. We declare that the invention was reduced to practice before February 21, 2002, in that the Source Code disclosed in Exhibit B was tested and found to run successfully before the effective date of Kim.

6. Additionally, we refer to a series of internal e-mails between the inventors and the programmer or between members of the Assignee discussing aspects of the invention, including the Universal Resource Locator (URL) redirect function (attached hereto as Exhibit C1-C5, collectively "Exhibit C"). We declare that these documents, Exhibit C, were created before February 21, 2002.

7. Claims 1-3 are pending in the application. With respect to the subject matter of the claims, the Source Code discloses an identification system to identify the plurality of content files and a relation system to relate related content files. *See*, Exhibit B, pages 1-7. The Source Code also discloses a query system to request content files from a server, a reference database to store information about the plurality of content files and a collection system to convert the plurality of content files into at least one collection file. *See*, Exhibit B, pages 7-

13. Additionally, the Source Code shows a conversion module to covert the at least one collection file into a master copy, wherein the master copy is in one of a plurality of formats. *See*, Exhibit B, pages 14-15. *See*, for example, the elements of claim 1.

8. Regarding the subject matter of the claims, the Source Code discloses the steps of, with respect to claim 3, transmitting a plurality of content files in various formats to an authoring system, and identifying, by the authoring system, the plurality of content files. *See*, Exhibit B, pages 1-5, and 14-15. The Source Code also discloses the steps of relating, by the authoring system, the plurality of content files and storing the related content files together. *See*, Exhibit B, pages 5-7 and 15-16.

9. Additionally, the Source Code discloses querying a server for additional content files. The code that queries a server for additional content files is disclosed in Exhibit B, page 7. The below code calls a subroutine that moves and extracts files from one location on the server to another location:

```
# extract the ecd template files
   print "Extracting DataPlay template archive ...", br;
   ExtractTemplateFiles ($tarRootPath, $dptartemplate);
```

The following subroutine, in Exhibit B, page 9, extracts a whole set of directories and files from a single archive and saves them to the server from which the code is being served:

```
sub ExtractTemplateFiles {
   my $tarRootPath = shift @_;
   my $tartemplate = shift @_;

   # chdir to the destination dir
   chdir($tarRootPath) or die "Can't chdir $tarRootPath: $!";
```

```
  # read the source tar file
#   print "about to read $tartemplate", br;
   my $tar  = new Archive::Tar($tartemplate);

   # write the source tar files to the new directory
   my @files = $tar->list_files;
   $tar->extract(@files);
}
```

Also see, Exhibit B, pages 7-9, specifically on page 8 note the subroutine identifiers for populating album and track directories.

10. Also, the Source Code supports populating a reference database with information related to the plurality of content files. *See*, Exhibit B, pages 9-12. Further, the Source Code discloses the steps of creating, by a user, at least one collection file including content files and producing, by the authoring system, a master copy of the collection file in a format specified by the user. The code in Exhibit B, pages 12-13 (subroutine CreateDataPlayOutput) creates and populates the file structure for a formatted disc, compiles that file structure into a single formatted file, and displays a link on a web page for the user to download the single file. Further, the code in Exhibit B, page 15 (subroutine CreateECDOutput) extracts a file structure from a single file, creates and saves a control file defining the content on the disc, then creates a single file with the file structure and control file. The subroutine then displays a link on a web page that the user can click to download the single file.

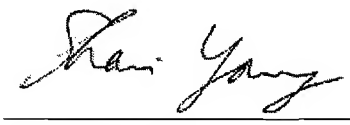
11. Exhibit C discloses a URL configuration system to identify a URL entered by a user into at least one collection file and to communicate with a redirect server to activate the URL, and

the steps of determining if a URL is in the collection file and communicating with a redirect server to activate the URL. Exhibits C1-C5 show the conception and development of the above concept. *See*, claims 1 and 3.

12. I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful statements and the like so made are punishable by fine or imprisonment or both under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

_____ _____
Jim BELCHER Dated

_____ _____
Howard SOROKA Dated



_____ 06/29/07
Shari YOUNG Dated

EXHIBIT A

EXHIBIT A

1. A system for remote authoring of content, comprising:

a plurality of content files; and

an authoring system storing the plurality of content files, comprising;

an identification system to identify the plurality of content files;

a relation system to relate related content files;

a query system to request content files from a server;

a reference database to store information about the plurality of content files;

a collection system to convert the plurality of content files into at least one collection file;

a conversion module to covert the at least one collection file into a master copy, wherein the master copy is in one of a plurality of formats; and

a URL configuration system to identify a URL entered by a user into the at least one collection file and to communicate with a redirect server to activate the URL.

2. The system of claim 1, further comprising a remote workstation for a user to access the authoring system.

3. A method of remotely authoring content, comprising:

transmitting a plurality of content files in various formats to an authoring system;

identifying, by the authoring system, the plurality of content files;

relating, by the authoring system, the plurality of content files;

storing the related content files together;

querying a server for additional content files;

populating a reference database with information related to the plurality of content files;

creating, by a user, at least one collection file including content files;

producing, by the authoring system, a master copy of the collection file in a format specified by the user;

determining if a URL is in the collection file;

communicating with a redirect server to activate the URL.

EXHIBIT B

EXHIBIT B

```
####################
# Source File Header
#!/usr/bin/perl
# !perl -wl015
#
# UMGMULTIMASTER.CGI - a script for building UMG ECD's and DataPlay discs
# AUTHOR: [redacted]
# CREATED: [redacted]
####################
#
# sub InputIDData - Input the project ID data
# passed: nothing
# returns: nothing
#
sub InputIDData {
my %stepDesc = @_;
my $pageTitle = "UMG Multimedia Pre-Mastering Database: Input ID Data";
my $instructions = "You must fill out all fields with a yellow background. All other fields (blue \
background) are optional. Please enter identifying data about this ECD.";


        # display form for ECD data input

        StartPage ($pageTitle, $instructions, %stepDesc);
        print start_form;

        PrintInputDataTable ($EDIT_FIELDS,
                             $TABLE_DATA{'userdata'}{'tableprompt'},
                             @{$TABLE_DATA{'userdata'}{'datafields'}});

        # print the album data input table
        PrintInputDataTable ($EDIT_FIELDS,
                             $TABLE_DATA{'projectinfo'}{'tableprompt'},
                             @{$TABLE_DATA{'projectinfo'}{'datafields'}});

        # prompt for multimedia output type (ECD, DataPlay, or both)
        print "<P><B><FONT SIZE=+1>$USER_INPUT_DATA{'outputmedia'}{'prompt'}</FONT>";

        print table ({ -bgcolor => $yellowbg,
                       -width => '600'},
                    Tr ({-align => 'CENTER'},
                       td (radio_group (-name=>'outputmedia',
                          -Values=>['ecdmediaout','dpmediaout', 'ecddpmediaout'],
                             -labels=>{ ecdmediaout=>"Enhanced CD",
                                        dpmediaout=>"DataPlay",
                                        ecddpmediaout=>"Both"},
                             -cols=>1,
                             -default => $USER_INPUT_DATA{'outputmedia'}{'value'},
                                                                          ))));

        # in order to save state, print all of the entry data that isn't
        # editable as hidden fields
        PrintHiddenECDDataValues (@{$stepDesc{'stepfields'}});
```

```
        print br,
            hidden (-name=>'currstep', -value=>'inputIDData', -force=>1),
            submit (-name=>'action',    -value => '<- Previous'),
                    submit (-name=>'action',   -value => 'Next ->'),
                    end_form;

        EndPage();
}

#
# sub InitInputTableData - initialize the descriptions of all input tables
#
sub InitInputTableDataAndStepDesc {

# first, the static tables

        %TABLE_DATA = (
                userdata => {
                datafields => ['userfirstname',
                                'userlastname',
                                'userphone',
                                'useremail'],
                   tableprompt => "Please Tell Us Who You Are:",
                },
                projectinfo => {
                        datafields => ['artistname',
                          'albumname',
                          'producer',
                          'recordlabel',
                          'cdname',
                          'selectionno',
                          'upcno'],
                    tableprompt => "Project Information:",
                },
                platformindependentfields => {
                        datafields => ['nsongs',
                          'nvideos',
                    'naudioclips',
                    'nurls',
                    'nemails',
                    'nlocalurls',
                    'nbookpages'],
               tableprompt => "Platform-Independent Content:",
             },
                webpagefields => {
                        datafields => ['webtitle0',
                                                        'weburl0'],
                        tableprompt => "Minimum Web Page Information:",
                },
                coverimagefilefields => {
                        datafields => ['frontcoversm',
                                                        'frontcovermed',
                                                        'frontcoverlg'],
                        tableprompt => "Cover (jacket) Image File Names:",
                },
```

```
        splashscreendata => {
                datafields => ['splashscreenurlname',
                        'splashscreenurl'],
                tableprompt => "Splash Screen Data:",
        },
        macspecificfields => {
                datafields => ['nmacapps',
                                                'nmacexecs',
                                                'nmacaudioclips',
                                                'nstxtfiles'],
                tableprompt => "Mac-Specific Content:",
        },
        pcspecificfields => {
                datafields => ['npcapps',
                                                'npcexecs',
                                                'npcaudioclips',
                                                'nrtffiles'],
                tableprompt => "Windows-Specific Content:",
        },
        dpspecificfields => {
                datafields => ['dpEncodingFormat'],
                tableprompt => "DataPlay Content:",
        },
);

# now, add the dynamic tables based on user input data
# the platform independent table data
my @platIndFields =         BuildDynamicTableData ('platformindependentfields');

my @macSpecFields = BuildDynamicTableData ('macspecificfields');

my @pcSpecFields  = BuildDynamicTableData ('pcspecificfields');

# now, initialize the list of fields specified for each step
# the fields for id data tables
my @idDataFields;
push @idDataFields, @{$TABLE_DATA{'userdata'}{'datafields'}};
push @idDataFields, @{$TABLE_DATA{'projectinfo'}{'datafields'}};
push @idDataFields, "outputmedia";

# the fields for content description tables
my  @contentDescFields;
push @contentDescFields, @{$TABLE_DATA{'platformindependentfields'}{'datafields'}};
push @contentDescFields, @{$TABLE_DATA{'coverimagefilefields'}{'datafields'}};
push @contentDescFields, 'mmcontentdesc';

my  @ecdSpecificFields;
push @ecdSpecificFields, @{$TABLE_DATA{'webpagefields'}{'datafields'}};
push @ecdSpecificFields, @{$TABLE_DATA{'macspecificfields'}{'datafields'}};
push @ecdSpecificFields, @{$TABLE_DATA{'pcspecificfields'}{'datafields'}};
push @ecdSpecificFields, 'splashscreen';
push @ecdSpecificFields, @{$TABLE_DATA{'splashscreendata'}{'datafields'}};

my  @dpSpecFields;
push @dpSpecFields, @{$TABLE_DATA{'dpspecificfields'}{'datafields'}};
```

```
%STEP_DESC = (
        selectAlbum => {
                stepnum => 0,
                steptitle  => "Choose Album",
                stepfields => [],
                stepfunc  => \&SelectAlbum,
                nextstep => 'inputIDData',
                skipthisstep => $FALSE,
        },
        inputIDData => {
                stepnum => 1,
                steptitle  => "Input Project ID Data",
                stepfields => \@idDataFields,
                stepfunc  => \&InputIDData,
                nextstep => 'inputContentDesc',
                prevstep => 'selectAlbum',
                skipthisstep => $FALSE,
        },
        inputContentDesc => {
                stepnum => 2,
                steptitle  => "Input Album Description",
                stepfields => \@contentDescFields,
                stepfunc  => \&InputContentDesc,
                nextstep => 'inputPlatformIndependentData',
                prevstep => 'inputIDData',
                skipthisstep => $FALSE,
        },
        inputPlatformIndependentData => {
                stepnum => 3,
                steptitle  => "Input Album Content",
                stepfields => \@platIndFields,
                stepfunc  => \&InputPlatformIndependentData,
                nextstep => 'inputECDSpecificData',
                prevstep => 'inputContentDesc',
                skipthisstep => $FALSE,
        },
        inputECDSpecificData => {
                stepnum => 4,
                steptitle  => "Input ECD Specific Data",
                stepfields => \@ecdSpecificFields,
                stepfunc  => \&InputECDSpecificData,
                nextstep => 'inputMacSpecificData',
                prevstep => 'inputPlatformIndependentData',
                skipthisstep => $FALSE,
        },
        inputMacSpecificData => {
                stepnum => 5,
                steptitle  => "Input Mac Specific Data",
                stepfields => \@macSpecFields,
                stepfunc  => \&InputMacSpecificData,
                nextstep => 'inputPCSpecificData',
                prevstep => 'inputECDSpecificData',
                skipthisstep => $FALSE,
        },
        inputPCSpecificData => {
                stepnum => 6,
```

```
                steptitle  => "Input PC Specific Data",
                stepfields => \@pcSpecFields,
                stepfunc  => \&InputPCSpecificData,
                nextstep => 'inputDPSpecificData',
                prevstep => 'inputMacSpecificData',
                skipthisstep => $FALSE,
        },
        inputDPSpecificData => {
                stepnum => 7,
                steptitle  => "Input DataPlay Specific Data",
                stepfields => \@dpSpecFields,
                stepfunc  => \&InputDataPlaySpecificData,
                nextstep => 'verifyData',
                prevstep => 'inputPCSpecificData',
                skipthisstep => $FALSE,
        },
        verifyData => {
                stepnum => 8,
                steptitle  => "Verify Data",
                stepfields => ['mmcontentdesc'],
                stepfunc  => \&VerifyData,
                nextstep => 'displayInstructions',
                prevstep => 'inputDPSpecificData',
                skipthisstep => $FALSE,
        },
        displayInstructions => {
                stepnum => 9,
                steptitle  => "File Creation Instructions",
                stepfields => [],
                stepfunc  => \&DisplayInstructions,
                nextstep => 'outputFiles',
                prevstep => 'verifyData',
                skipthisstep => $FALSE,
        },
        outputFiles => {
                stepnum => 10,
                steptitle  => "Create Output Files",
                stepfields => [],
                stepfunc  => \&OutputFiles,
                prevstep => 'displayInstructions',
                skipthisstep => $FALSE,
        },
    );

    # set the appropriate "skipthisstep" flags
    FindSkippedSteps ();
}

###################
```

```
###################
# Source File Header
#!/usr/bin/perl
# !perl -wl015
#
# UMGMULTIMASTER.CGI - a script for building UMG ECD's and DataPlay discs
# AUTHOR: [redacted]
# CREATED: [redacted]
###################
#
# sub SaveEntryData - Save the field data values to a text file
# passed: nothing
# returns: nothing
#
sub SaveEntryData {
        my %dataValues;
        # selection number must be defined since it's used as the
        # hash for each ecd's data
        if (defined $USER_INPUT_DATA{'cdname'}{'value'}) {
                # copy the field data values into a hash of their own
                for (keys %USER_INPUT_DATA) {
                        $dataValues{$_} = $USER_INPUT_DATA{$_}{'value'};
                }
#               UMGMMDataDumper::SetUMGMMInputFile  ($datafilename);
#               UMGMMDataDumper::SetUMGMMOutputFile ($datafilename);
#               UMGMMDataDumper::WriteUMGMMData(%dataValues);
                UMGMMDBManager::UMDBSetDBParams ($dbname, $dbUserName, $dbPassword);
                UMGMMDBManager::WriteUMGMMData (%dataValues);
        }
}

###################
# Source File Header
#
# UMGMMDataManager.pm - data management routines for UMGMultiMaster tool. This version uses
#               the Data::Dumper perl module to store the data in a flat file.
# AUTHOR: [redacted]
# CREATED: [redacted]
# MODIFIED: [redacted] - changed from ECDDataManger to UMGMMDataManager
#
#
###################
# sub WriteUMGMMData - write data for selected disc
#               current implementation reads all data, adds/modifies data
#               for selected disc, then writes all data
# passed: umgmm data hash to be written
# returns: nothing
#
sub WriteUMGMMData {
        my %selectionData = @_;

        # read  %allUMGMMs from file
        my $allUMGMMDataRef = ReadAllUMGMMData ();
        my %allDiscs = %$allUMGMMDataRef;
```

```
#       PrintAllUMGMMData (%allUMGMMData);

        # copy this disc's data to the allUMGMM hash
        # for now, use the selecton number as the hash key
        my $cdname = $selectionData{"cdname"};
        my $umgmmKey;

        $allDiscs {$cdname} = {};
        for $umgmmKey (keys %selectionData) {
                $allDiscs {$cdname} {$umgmmKey} = $selectionData{$umgmmKey};
                DebugPrint ("setting allUMGMMData \{$cdname\} \{$umgmmKey\} to
$selectionData{$umgmmKey}");
        }

        # write the new file
        # NOTE: add file locking here!
        open (FILE, ">$umgmmOutputFile") or die "Can't open $umgmmOutputFile: $!";
        print FILE Data::Dumper->Dump ([\%allDiscs], ['*allDiscs']);
        close FILE;

}



##################
# Source File Header
#!/usr/bin/perl
# !perl -wl015
#
# UMGMULTIMASTER.CGI - a script for building UMG ECD's and DataPlay discs
# AUTHOR: [redacted]
# CREATED: [redacted]
##################


sub CreateDataPlayOutput {

        # untar the dataplay template directories
        my $tarRootPath = catfile ($fileoutputdir, $USER_INPUT_DATA{'cdname'}{'value'});

        # extract the ecd template files
        print "Extracting DataPlay template archive ...", br;
        ExtractTemplateFiles ($tarRootPath, $dptartemplate);

        # rename the extracted directory tree to the name of this cd
        chdir($tarRootPath) or die "Can't chdir $tarRootPath: $!";
        my $dpDir = $USER_INPUT_DATA{'cdname'}{'value'}."DP";
        rename "dpFileTemplate", $dpDir;

        # write Contents.ddl to root/"Content Manager" directory
        my $contentDir = catfile ($tarRootPath, $dpDir, "Content Manager");
        if (! -e ($contentDir)) {
                mkdir $contentDir, 0777 or die "Can't mkdir $contentDir: $!";
        }

        my $contentsDDLTemplate = catfile ($fileinputdir, "Contents.ddl.tmpl");
```

```
	my $outputFileName = catfile ($contentDir, "Contents.ddl");
	FillAndWriteTemplateFile ($contentsDDLTemplate, $outputFileName);


	# populate root/Music/_Playlists_
	my $musicDir = catfile ($tarRootPath, $dpDir, "Music");
	if (! -e ($musicDir)) {
		mkdir $musicDir, 0777 or die "Can't mkdir $musicDir: $!";
	}
	my $playlistDir = catfile ($musicDir, "_Playlists_");
	if (! -e ($playlistDir)) {
		mkdir $playlistDir, 0777 or die "Can't mkdir $playlistDir: $!";
	}
	chdir($playlistDir) or die "Can't chdir $playlistDir: $!";

	my $defaultPlaylistTemplate = catfile ($fileinputdir, "Default.playlist.tmpl");
	my $outputFileName = catfile ($playlistDir, "Default.playlist");
	FillAndWriteTemplateFile ($defaultPlaylistTemplate, $outputFileName);

	# create the album directory under root/Music
	chdir($musicDir) or die "Can't chdir $musicDir: $!";
	my $albumDir = catfile ($musicDir, $USER_INPUT_DATA{'albumname'}{'value'});
	if (! -e ($albumDir)) {
		mkdir $albumDir, 0777 or die "Can't mkdir $albumDir: $!";
	}


	# populate the album directory with "Thumbnail.jpg" and album-
	# specific files (video, audio, text, images, credits, etc.)

	# create the track directories under root/Music/Album Title
	chdir($albumDir) or die "Can't chdir $albumDir: $!";
	for (my $i=0; $i<$USER_INPUT_DATA{'nsongs'}{'value'}; $i++) {
		my $songKey = "songname$i";
		my $songDir = $USER_INPUT_DATA{$songKey}{'value'};
		if (! -e ($songDir)) {
			mkdir $songDir, 0777 or die "Can't mkdir $songDir: $!";
		}
	}

	# populate the track directories with "Thumbnail.jpg" for each track
	# populate the track directories with track-specific files (lyrics, credits, etc.)

	print "Writing compressed DataPlay output file ...", br;
	WriteTarFile ($tarRootPath, $dpDir);

	print a ({-href=>"$ftpdir/$dpDir/$dpDir.tar"},
		"Click Here to download the compressed DataPlay file for:
		 $USER_INPUT_DATA{'albumname'}{'value'}"), br;

}

sub CreateECDOutput {

	my $tarRootPath = catfile ($fileoutputdir, $USER_INPUT_DATA{'cdname'}{'value'});
```

```
        # extract the ecd template files
        print "Extracting ECD template archive ...", br;
        ExtractTemplateFiles ($tarRootPath, $ecdtartemplate);

        # rename the extracted directory tree to the name of this cd
        chdir($tarRootPath) or die "Can't chdir $tarRootPath: $!";
        my $ecdDir = $USER_INPUT_DATA{'cdname'}{'value'}."ECD";
        rename "ecdFileTemplate", $ecdDir;

        print "Creating Script files ...", br;
        CreateDCDLFile ($tarRootPath, $ecdDir);

        print "Writing compressed ECD output file ...", br;
        WriteTarFile ($tarRootPath, $ecdDir);

        print a ({-href=>"$ftpdir/$ecdDir/$ecdDir.tar"},
                "Click Here to download the compressed ECD file for:
                 $USER_INPUT_DATA{'albumname'}{'value'}"), br;


}

sub ExtractTemplateFiles {
        my $tarRootPath = shift @_;
        my $tartemplate = shift @_;

        # chdir to the destination dir
        chdir($tarRootPath) or die "Can't chdir $tarRootPath: $!";

        # read the source tar file
#       print "about to read $tartemplate", br;
        my $tar  = new Archive::Tar($tartemplate);

        # write the source tar files to the new directory
        my @files = $tar->list_files;
        $tar->extract(@files);
}
################

###################
# Source File Header
#!/usr/bin/perl
# !perl -wl015
#
# umgmmDBManager.pm - module for managing reading & writing data to the umgmultimaster DB
# AUTHOR: [redacted]
# CREATED: [redacted]
###################

#
# sub WriteUMGMMData - write the UMGMMData hash values to the appropriate tables in the database
#                                                 by executing either a CREATE or an UPDATE
#
sub WriteUMGMMData {
        my %selectionData = @_;
```

```
	my $albumId = $selectionData{'albumid'};
	my $userId  = $selectionData{'userid'};

	# connect to the DB, if not already connected
	UMDBConnect();

	if ($albumId) {
		# print "updating data for album $selectionData{albumname} $albumId user $userId\n";
		UMDBUpdateEntry (\%selectionData, $albumId, $userId);
	} else {
		# print "creating data for album $selectionData{albumname}\n";
		UMDBCreateEntry (\%selectionData);
	}
}
sub UMDBCreateEntry {
	my $selDataRef = shift @_;

	my %selectionData = %$selDataRef;

	my $albumname = $dbh->quote($selectionData{'albumname'});
#	print "Create new album entry for $albumname\n";

	my $userId = UMGMMUserAuth::UMUAGetCurrentUserID ();

	# write the album table data
	my $albumId = UMDBCreateAlbumRow (\%selectionData, $userId);

	# update the usereditable table (row created when db created)
	UMDBUpdateTableRow ($selDataRef, "user_id", $userId, "user_editable");

	# write the assest description data to the appropriate tables
	UMDBCreateAssetData (\%selectionData, $albumId);
}
#
# sub UMDBCreateAlbumRow - creates the album table row
#
sub UMDBCreateAlbumRow {
	my $selectionDataRef = shift @_;
	my $userId	= shift @_;

	my %selectionData = %$selectionDataRef;

	# first, build the list of data values that are in %selectionData
	my @tableColumnNames = UMDBGetNonIDColumnNames ("album");
	my @tableColumnValues;
	my $selectionDataQuoted;

	foreach my $colName (@tableColumnNames) {
		$selectionDataQuoted = $dbh->quote($selectionData{$colName});
		push @tableColumnValues, $selectionDataQuoted;
	}

	# add the user_id column
	push @tableColumnNames, "user_id";
	push @tableColumnValues, $userId;
```

```
        # do the insert
        my $albumId = UMDBInsert ("album", \@tableColumnNames, \@tableColumnValues);
}

#
# sub UMDBCreateAssetData - write asset property data to an asset property
#                           table
#
sub UMDBCreateAssetData {
        my $selDataRef = shift @_;
        my $albumId    = shift @_;

        # first, read the list of asset count fields and asset data table names
        #  (e.g. 'nsongs', 'songdatatable')
        my $sth = $dbh->prepare
                ("select assetcountfieldname, assetpropertytablename from asset_property_table_def");
        $sth->execute();

        my $assetMapRefs = $sth->fetchall_arrayref({assetcountfieldname=>1, assetpropertytablename=>1});

        # now write the data for asset property each table
        foreach my $assetTableDefRef (@$assetMapRefs){
                UMDBCreateAssetPropertyTableRows ($selDataRef,

        $assetTableDefRef,

        $albumId);
        }
}

#
# sub UMDBCreateAssetPropertyTableRows - write asset property data to an asset property
#                           table
#
sub UMDBCreateAssetPropertyTableRows {
        my $selectionDataRef = shift @_;
        my $assetTableDefRef = shift @_;
        my $albumId          = shift @_;

        my %selectionData = %$selectionDataRef;


        my $countFieldName = $assetTableDefRef->{'assetcountfieldname'};
        my $tableName      = $assetTableDefRef->{'assetpropertytablename'};

        # if there are no assets of this type, do nothing
        if (!$selectionData{$countFieldName}) { return; }

        # get the field names for this asset property table
        my @colNames = UMDBGetNonIDColumnNames ($tableName);

        # add album_id to colNames
        push @colNames, "album_id";
        my @placeHolders = ("?") x @colNames;   # a list of ?'s corresponding to field names

        # prepare the insert statement for all rows
```

```
        my $sqlStatement = sprintf "INSERT INTO %s (%s) VALUES (%s)",
                $tableName, join(",", @colNames), join (",", @placeHolders);

#       print "about to write to table: $tableName\n";
#       print "sqlStatement: $sqlStatement\n";

        my $sth = $dbh->prepare ($sqlStatement);

        # now do the inserts, one row for each asset
        # (i.e. nsongs rows inserted into songdata table)
        for (my $i=0; $i < $selectionData{$countFieldName}; $i++) {
                my @columnValues;
                foreach my $colName (@colNames) {
                        my $selectionValue;
                        if ($colName eq "album_id") {
                                $selectionValue = $albumId;
                        } else {
                                # build the hash name based on count (i.e. songname1, songname2, etc.)
                                my $hashName = $colName.$i;
                                $selectionValue = $selectionData{$hashName};
                        }
                        push @columnValues, $selectionValue;
                }
                # execute the cmd, binding the values at execute time
#               print "about to execute $sqlStatement\n";
#               print "with values: @columnValues\n";
                $sth->execute (@columnValues);
        }



###################
# Source File Header
#!/usr/bin/perl
# !perl -wl015
#
# UMGMULTIMASTER.CGI - a script for building UMG ECD's and DataPlay discs
# AUTHOR: [redacted]
# CREATED: [redacted]
###################
sub CreateDataPlayOutput {

        # untar the dataplay template directories
        my $tarRootPath = catfile ($fileoutputdir, $USER_INPUT_DATA{'cdname'}{'value'});

        # extract the ecd template files
        print "Extracting DataPlay template archive ...", br;
        ExtractTemplateFiles ($tarRootPath, $dptartemplate);

        # rename the extracted directory tree to the name of this cd
        chdir($tarRootPath) or die "Can't chdir $tarRootPath: $!";
        my $dpDir = $USER_INPUT_DATA{'cdname'}{'value'}."DP";
        rename "dpFileTemplate", $dpDir;

        # write Contents.ddl to root/"Content Manager" directory
        my $contentDir = catfile ($tarRootPath, $dpDir, "Content Manager");
```

```
if (! -e ($contentDir)) {
        mkdir $contentDir, 0777 or die "Can't mkdir $contentDir: $!";
}

my $contentsDDLTemplate = catfile ($fileinputdir, "Contents.ddl.tmpl");
my $outputFileName = catfile ($contentDir, "Contents.ddl");
FillAndWriteTemplateFile ($contentsDDLTemplate, $outputFileName);


# populate root/Music/_Playlists_
my $musicDir = catfile ($tarRootPath, $dpDir, "Music");
if (! -e ($musicDir)) {
        mkdir $musicDir, 0777 or die "Can't mkdir $musicDir: $!";
}
my $playlistDir = catfile ($musicDir, "_Playlists_");
if (! -e ($playlistDir)) {
        mkdir $playlistDir, 0777 or die "Can't mkdir $playlistDir: $!";
}
chdir($playlistDir) or die "Can't chdir $playlistDir: $!";

my $defaultPlaylistTemplate = catfile ($fileinputdir, "Default.playlist.tmpl");
my $outputFileName = catfile ($playlistDir, "Default.playlist");
FillAndWriteTemplateFile ($defaultPlaylistTemplate, $outputFileName);

# create the album directory under root/Music
chdir($musicDir) or die "Can't chdir $musicDir: $!";
my $albumDir = catfile ($musicDir, $USER_INPUT_DATA{'albumname'}{'value'});
if (! -e ($albumDir)) {
        mkdir $albumDir, 0777 or die "Can't mkdir $albumDir: $!";
}


# populate the album directory with "Thumbnail.jpg" and album-
# specific files (video, audio, text, images, credits, etc.)

# create the track directories under root/Music/Album Title
chdir($albumDir) or die "Can't chdir $albumDir: $!";
for (my $i=0; $i<$USER_INPUT_DATA{'nsongs'}{'value'}; $i++) {
        my $songKey = "songname$i";
        my $songDir = $USER_INPUT_DATA{$songKey}{'value'};
        if (! -e ($songDir)) {
                mkdir $songDir, 0777 or die "Can't mkdir $songDir: $!";
        }
}

# populate the track directories with "Thumbnail.jpg" for each track
# populate the track directories with track-specific files (lyrics, credits, etc.)

print "Writing compressed DataPlay output file ...", br;
WriteTarFile ($tarRootPath, $dpDir);

print a ({-href=>"$ftpdir/$dpDir/$dpDir.tar"},
      "Click Here to download the compressed DataPlay file for:
       $USER_INPUT_DATA{'albumname'}{'value'}"), br;

}
```

```
##################
# Source File Header
#!/usr/bin/perl
# !perl -wl015
#
# UMGMULTIMASTER.CGI - a script for building UMG ECD's and DataPlay discs
# AUTHOR: [redacted]
# CREATED: [redacted]
##################
#
# sub OutputFiles - Create the appropriate out files, including:
#               - untarring the template directory structure (for ECD's)
#               - creating the dcdl file (for ECD's)
#               - untarring  the DataPlay dir structure
#               - creating relevent DataPlay files
# passed: nothing
# returns: nothing
#
sub OutputFiles {
my %stepDesc = @_;
my $pageTitle = "UMG Multimedia Pre-Mastering Database: Output Files";
my $instructions = "The compressed ECD file has been created. It is in zipped tar
format. To download the file, click on the link below.";


        StartPage ($pageTitle, $instructions, %stepDesc);
        h1 ($pageTitle);

        my $tarRootPath = catfile ($fileoutputdir, $USER_INPUT_DATA{'cdname'}{'value'});

        # if the output directory already exists, delete it and start over
        if (-e ($tarRootPath)) {
           print "Removing previous version of $USER_INPUT_DATA{'cdname'}{'value'} Discs ... ", br;
           rmtree ($tarRootPath, 0, 1) or die "Can't rmtree pre-existing $tarRootPath: $!";
        }
        mkdir $tarRootPath, 0777 or die "Can't mkdir $tarRootPath: $!";

#       The old way!
#       if (! -e ($tarRootPath)) {
#               mkdir $tarRootPath, 0777 or die "Can't mkdir $tarRootPath: $!";
#       }

        if (($USER_INPUT_DATA{'outputmedia'}{'value'} eq 'ecdmediaout') ||
                ($USER_INPUT_DATA{'outputmedia'}{'value'} eq 'ecddpmediaout')) {
                        CreateECDOutput ();
        }

        if (($USER_INPUT_DATA{'outputmedia'}{'value'} eq 'dpmediaout') ||
                ($USER_INPUT_DATA{'outputmedia'}{'value'} eq 'ecddpmediaout')) {
                        CreateDataPlayOutput ();
        }


        print start_form;
        PrintHiddenECDDataValues (@{$stepDesc{'stepfields'}});
```

```
        print br,
            hidden (-name=>'prevstep', -value=>$stepDesc{'prevstep'}, -force=>1),
            hidden (-name=>'currstep', -value=>'outputFiles', -force=>1),
            submit (-name=>'action',   -value => '<- Previous'),
                    end_form;

        EndPage();

        # save the USER_INPUT_DATA values to a file for future use
        SaveEntryData ();
}
sub CreateECDOutput {

        my $tarRootPath = catfile ($fileoutputdir, $USER_INPUT_DATA{'cdname'}{'value'});

        # extract the ecd template files
        print "Extracting ECD template archive ...", br;
        ExtractTemplateFiles ($tarRootPath, $ecdtartemplate);

        # rename the extracted directory tree to the name of this cd
        chdir($tarRootPath) or die "Can't chdir $tarRootPath: $!";
        my $ecdDir = $USER_INPUT_DATA{'cdname'}{'value'}."ECD";
        rename "ecdFileTemplate", $ecdDir;

        print "Creating Script files ...", br;
        CreateDCDLFile ($tarRootPath, $ecdDir);

        print "Writing compressed ECD output file ...", br;
        WriteTarFile ($tarRootPath, $ecdDir);

        print a ({-href=>"$ftpdir/$ecdDir/$ecdDir.tar"},
              "Click Here to download the compressed ECD file for:
               $USER_INPUT_DATA{'albumname'}{'value'}"), br;


}

        ###################
        # Source File Header
        #!/usr/bin/perl
        # !perl -wl015
        #
        # UMGMULTIMASTER.CGI - a script for building UMG ECD's and DataPlay discs
        # AUTHOR: [redacted]
        # CREATED: [redacted]
        ###################

        # populate the album directory with "Thumbnail.jpg" and album-
        # specific files (video, audio, text, images, credits, etc.)

        # create the track directories under root/Music/Album Title
        chdir($albumDir) or die "Can't chdir $albumDir: $!";
        for (my $i=0; $i<$USER_INPUT_DATA{'nsongs'}{'value'}; $i++) {
                my $songKey = "songname$i";
                my $songDir = $USER_INPUT_DATA{$songKey}{'value'};
                if (! -e ($songDir)) {
```

```
                mkdir $songDir, 0777 or die "Can't mkdir $songDir: $!";
        }
}

# populate the track directories with "Thumbnail.jpg" for each track
# populate the track directories with track-specific files (lyrics, credits, etc.)
```

EXHIBIT C

EXHIBIT C1

Forwarded Message
From: "[redacted]@umusic.com>
Date:
To: [redacted]@umusic.com>, "[redacted]@umusic.com>, "[redacted]@umusic.com>
Conversation: umusic.com
Subject: umusic.com

Right now the "umusic.com" domain actually redirects to the green UMG site, which is really located at www.universalstudios.com/music.

With new UMG portal, do we want to actually house the site at the /umusic directory on the server, and then redirect any hard coded references to "www.universalstudios.com/music" to the "www.umusic.com" site?
This is basically a "cleanup" issue, but wanted your thoughts as it might affect [redacted] ECD setup under the /umusic directory.

Thank you.

[redacted]@umusic.com

------ End of Forwarded Message

EXHIBIT C2

------ Forwarded Message
From: "[redacted]@umusic.com>
Date: [redacted]
To: "[redacted]@umusic.com>
Cc: "[redacted]@umusic.com>, "[redacted]
[redacted]@umusic.com>
Conversation: URL redirects for DVD-Audio
Subject: URL redirects for DVD-Audio

Dear [redacted],

One issue in DVD-Audio that keeps coming up between [redacted] and the labels is that of URLs on the disc.

The labels (specifically [redacted] so far) wants the URL to be their own URL visible on the disc. We have been arguing for a [redacted] redirect URL which will allow us to do two things:

1. Fix broken, dead, or hostile links in the future
2. Track DVD-A users who visit our URL redirect for the compilation of consumer data

Right now the discussion is confined to which URL appears on the main screen artwork in bitmapped form, but DVD-Audio also supports the embedding of URLs in the disc itself in special "slots" that may be read by future software and hardware players.

Of course if the URL on the disc is the label's, it is doubtful that we will be able to track any specific consumer data on DVD-A usage unless an effort is made by each label group to track that information and share it with us. (Note that with our URL we simply want to redirect the user to the label site, but also track the unique users.)

Without a specific corporate policy addressing this issue, it will continue to be a non-productive argument between us and the labels.

Could you escalate this up the ladder so we may get this resolved in the next few weeks? Thanks for all your help :-)

Regards,
[redacted]

[redacted]

------ End of Forwarded Message

EXHIBIT C3

------ Forwarded Message
From: "[redacted]@rosetta.com>
Date: [redacted]
To: [redacted]@yahoo.com>, "[redacted]@umusic.com"
<[redacted]@umusic.com>
Subject: Re: Mac Read Me

Here are a few thoughts. BTW, your pictures didn't make it through the email. They are in the resource fork of a SimpleText document, and whatever mailer you're using does not send the Mac-specific part of attached documents. Sadly, this also applies to the text styles, so I am not able to see which parts you italicized and underlined, and therefore this email doesn't address any problems there.

[redacted]

(1) System requirements. Software requirements are real, hardware requirements are my subjective feel and are negotiable.

Macintosh:
MacOS 8.6 or higher, but not MacOS X.
266 MHz PowerPC, 32 MB memory, 20 MB free disk space.

Windows:
Windows 95, 98, 2000, Me, or NT 4.0 or higher.
200 MHz Pentium, 64 MB memory, 20 MB free disk space.

(2) Well, UMP still needs the Internet Config API. For all the versions of MacOS we support, Internet Config shipped as a part of the OS and should already be there. We could either take it out of our installer completely, or leave it but only install it when no existing Internet Config is found (do we do that already?). We should certainly make sure not to overwrite an existing higher-version Internet Config with our own. We are shipping 1.4 and recent MacOS's shipped 2.0. I would vote for taking it out completely; if it turns up missing, that's just like any other broken part of the OS.

Also see (4) below.

(3) I think we should use http://www.apple.com/quicktime/ This is the top level of Apple's QuickTime site and its URL should be pretty stable. This leaves users to figure out for themselves

where to download, but practically all users should already have what they need -- suitable versions of QuickTime were part of every MacOS version we support. I think a more specific URL into the Apple site might become invalid when they rearrange things, and until UMG signs up to be custodian in perpetuity it's probably not wise to redirect through there.

However, we ought to test with QT 5.0 before we ship. [redacted], can you either do this or make sure f-test does?

(4) The Internet Config notes need updating. Modern MacOS's have an "Internet Assistant" and an "Internet" control panel of Apple's, which are supposed to configure the Internet Config settings at OS install time and to adjust those settings later, respectively. Users are no longer supposed to talk to IC itself.

(5) I imagine the issues in the troubleshooting section are still quite important to our users, but the specific path to new CD drivers and the cookbook Mac UI steps are probably out of date. In fact, the UI steps are probably different on the various MacOS versions we're supporting. I defer to [redacted] on whether there are additional troubleshooting issues for UMP 2.0.

[redacted]

At [redacted], [redacted] wrote:
> [redacted]
>
> I've done a first pass at the Mac readme file for UMP
> 2.0. I feel so proud that I was able to figure out
> how to put images in there. Sigh. I love the
> Internet.
>
> Anyway, I've italicized and underlined the parts that
> I didn't change because I believe it's wrong and I'm
> not sure what the correct information is. To
> summarize:
>
> 1. What are the minimum hardware requirements for Mac
> UMp?
>
> 2. Is internet config still required? (The installer
> still installs it.)
>
> 3. What URL should we point them to for Quicktime

> installs? [redacted], we had talked about setting up a
> redirect at universal so that it would always point to
> the most uptodate QT location.
>
> 4. Do we need all those 'Internet Config'notes?
>
> 5. Is that 'troubleshooting' section even relevent any
> more?
>
> I await your comments... Tomorrow, the PC readme. Oh
> joy!
>
> -[redacted]
>
> P.S. I'l put a copy of the read me on each of your ftp
> servers, just in case email munges it.
>
>
> _____
> Do You Yahoo!?
> Get personalized email addresses from Yahoo! Mail - only $35
> a year! http://personal.mail.yahoo.com/
> Content-Type: [redacted]"
> Content-Description: [redacted]
> Content-Disposition: attachment; filename="[redacted]"
>
> Attachment converted: [redacted]

------ End of Forwarded Message

EXHIBIT C4

------ Forwarded Message
From: [redacted]@yahoo.com>
Date: [redacted]
To: "[redacted]@umusic.com" <[redacted]@umusic.com>
Subject: Crossing i's and dotting t's

[redacted],

Here's a list of the last little things that I think
you need to do for this release:

1. Review evangelism disc for anything I forgot.

2. Create redirects/web pages for:
   www.umusic.com/evangelism/bbking -> www.bbking.com
   www.umusic.com/evangelism/authhoring -> ?

3. Update the www.umusic.com/ecdsupport website

4. Update [redacted] registration.

I think that's it for now.

Hope you're having a good trip... talk to you soon.

-[redacted]

Do You Yahoo!?
Make international calls for as low as $.04/minute with Yahoo! Messenger
http://phonecard.yahoo.com/

------ End of Forwarded Message

EXHIBIT C5

------ Forwarded Message
From: [redacted]@yahoo.com>
Date: [redacted]
To: "[redacted]@umusic.com" <[redacted]@umusic.com>,
"[redacted]@umusic.com" <[redacted]@umusic.com>
Subject: Pre-Holiday Status

Hey Guys,

I'm about to wind down and take some time over the next week (which I'll be making up after the first of the year, of course). Anyway, wanted to give you a quick summary of my status.

1. Project Manager Tool
- The version installed on Woodpecker allows you to select multiple locked/unlocked albums for a DataPlay disc. It is still only generating DPMMF for a single, unlocked disc. My dev versin is ALMOST generating DPMMF for multiple unlocked discs. The next step is to get it to generate DPMMF for multiple locked/unlocked discs.

- The following features are yet to be implemented:
  - 'Manufacturing Flag' - this will freeze the content for all albums and also generate the correct file handles, as opposed to the QDesign file handles that are generated for testing
  - Add user prompts for additional DPMMF and Gamma Disk fields described below
- move DataPlay fields (file encoding format) from Album Manager to Project Manager
- add VOC files and associated prompts if necessary

2. Album Manager Tool
  - Add a link back to the Project Manager Tool on the last step
  - Only display ECD-specific prompts to admin user (SplashScreen, Mac/PC content prompt)
- Get rid of DataPlay specific prompts (see above change to Project Manager)

3. Open Issues w/DataPlay
- We need to get back to [redacted] about what the

requirements are for specifying SelectionNumber, UPC, and SKU for albums, sides, & collections.
- Need to get more info on SidePartNumber, CollectionContentID, and writeable disk space for their GammaDisk delivery format.

4. Little ECD things
- I've got to get the real QUAC files from [redacted] and integrate the selection of the correct one based on # of tracks.
- We need to implement the URL redirect stuff.

5. Bugs, Testing, & Code Bloat
There are a bunch of little bugs that I know about and I'm sure there are tons of bugs that I don't know about. We just need to carve out some time for more extensive testing. Also, the code is getting a bit bloated and if the project really is going to continue for a longish period of time I'd like to do some re-organization. Kind of clean the closets, if you will. :)

6. Pimping it up
I'm still hoping we'll get a real UI/graphic designer to spend a couple of days making things look nicer.

Well, that's the scores and hilghts. You know where to find me with questions/answers/etc.

Happy Holidays, dudes.

-[redacted]

------ End of Forwarded Message